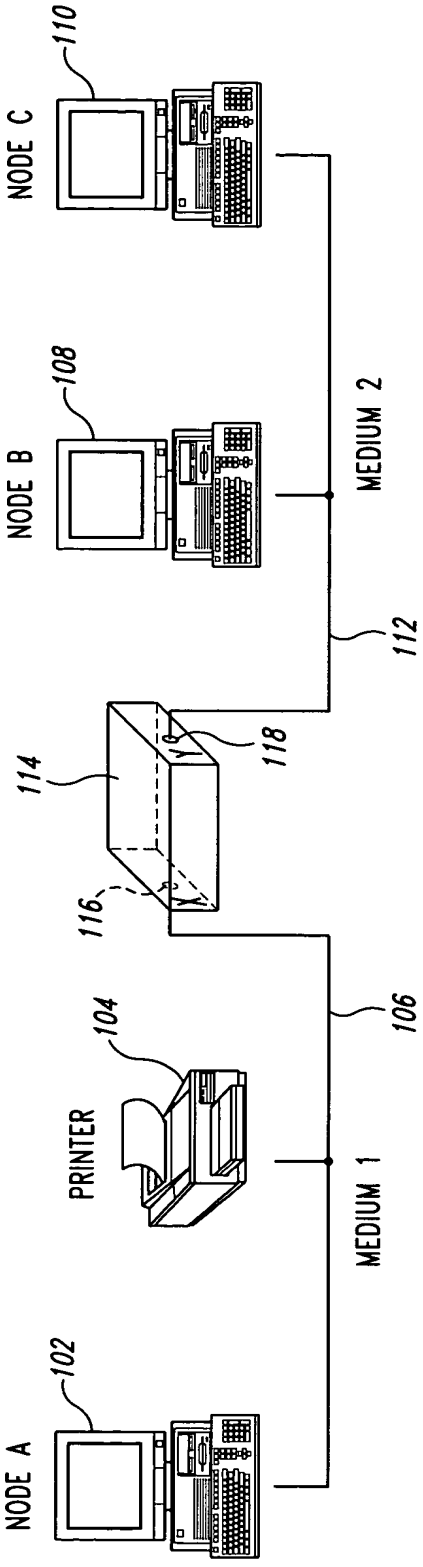
NODE C
110
NODE B
108
MEDIUM 2
112
114
118
116
106
PRINTER
104
MEDIUM 1
NODE A
102

Fig. 1

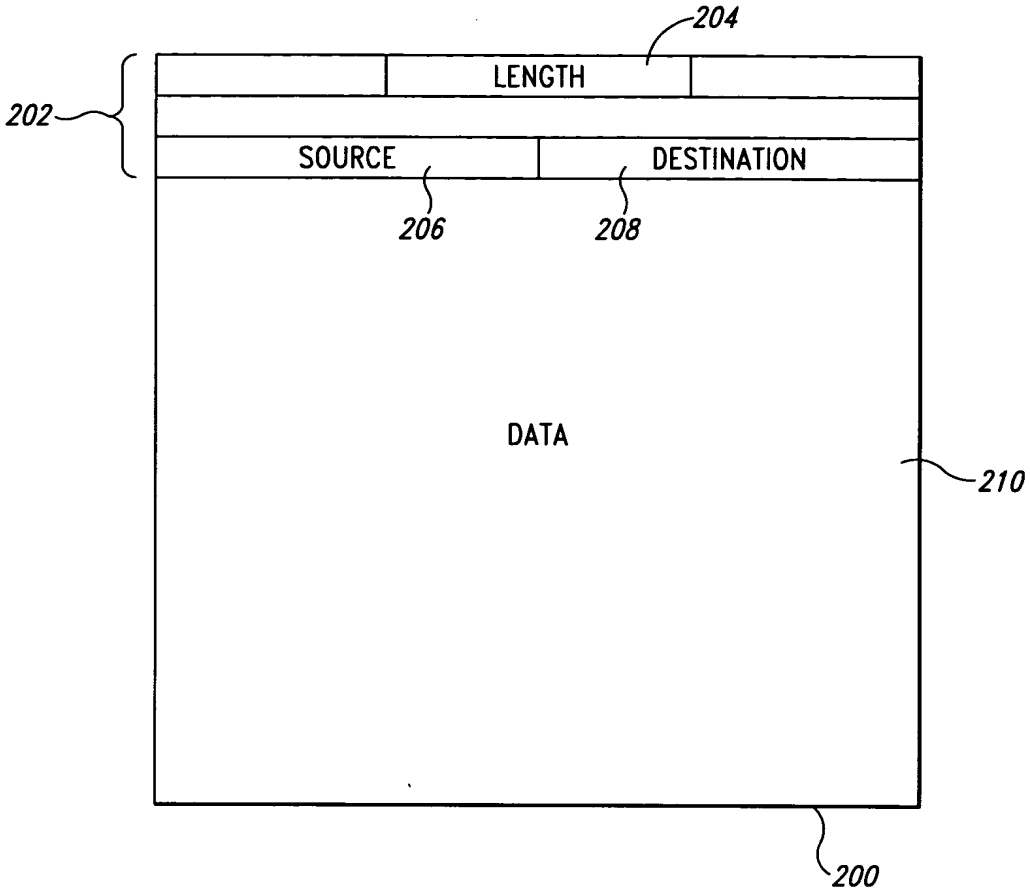

Fig. 2

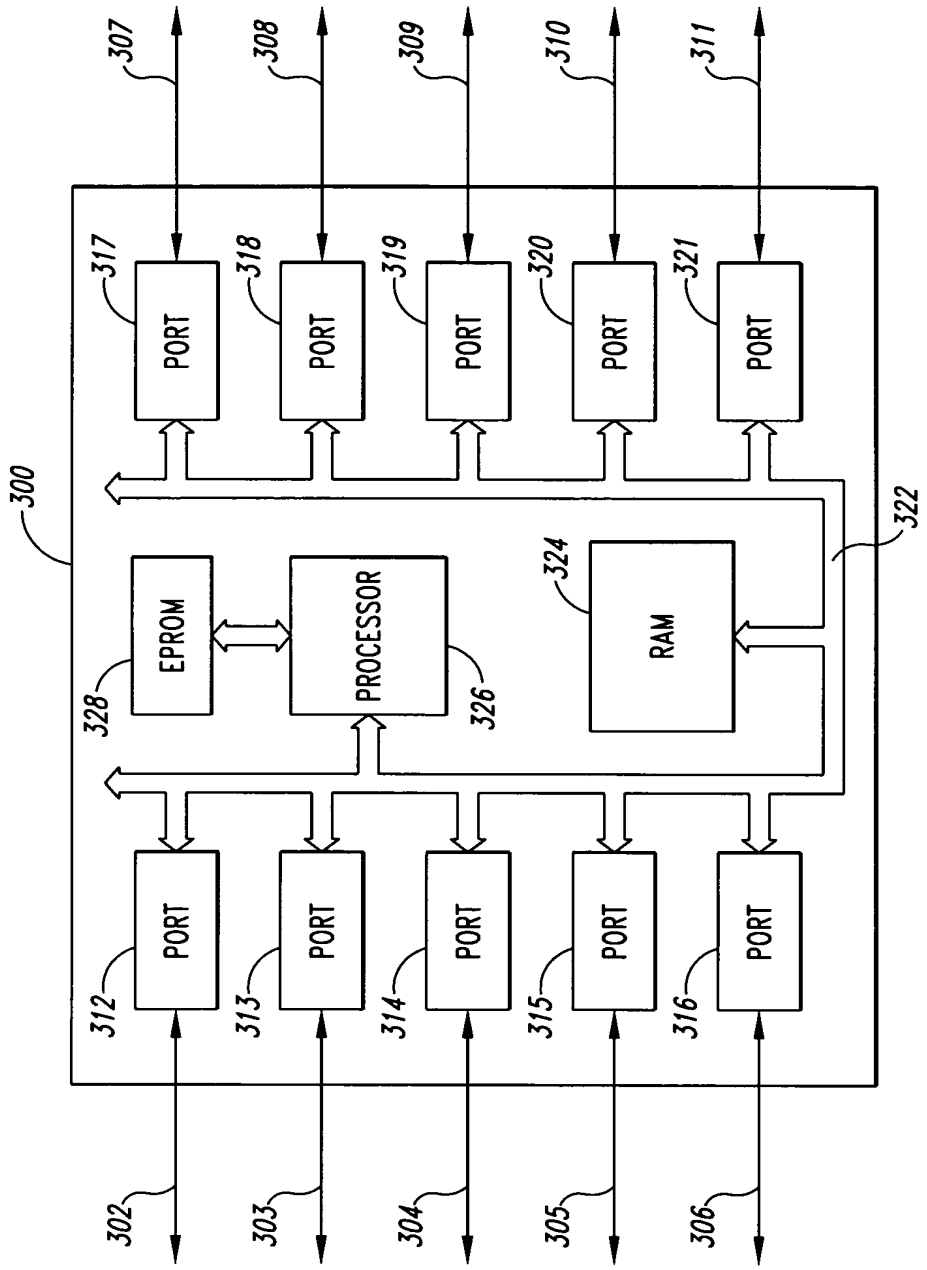
300
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
324
326
328
PORT
EPROM
PROCESSOR
RAM

Fig. 3

ADDRESS	PORT
NODE A	X
PRINTER	X
NODE B	Y
NODE C	Y

402 404

Fig. 4

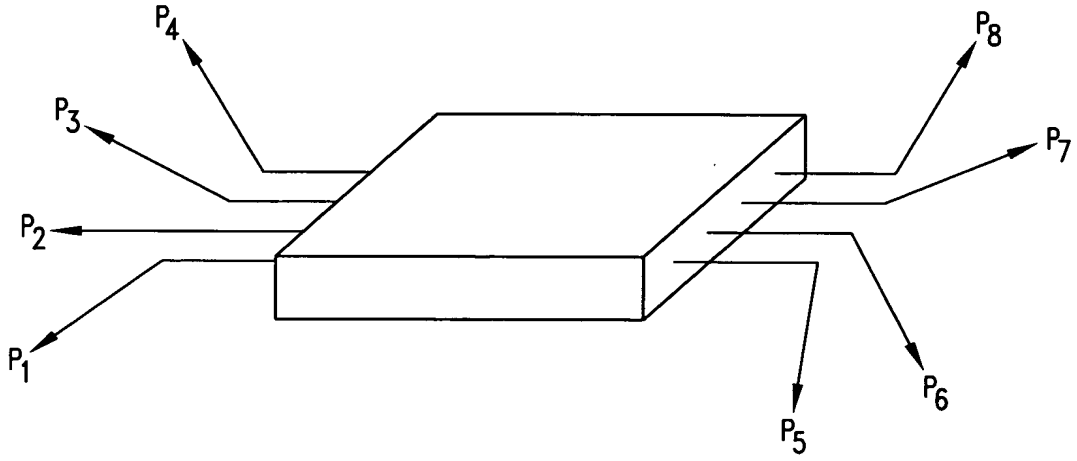

Fig. 5

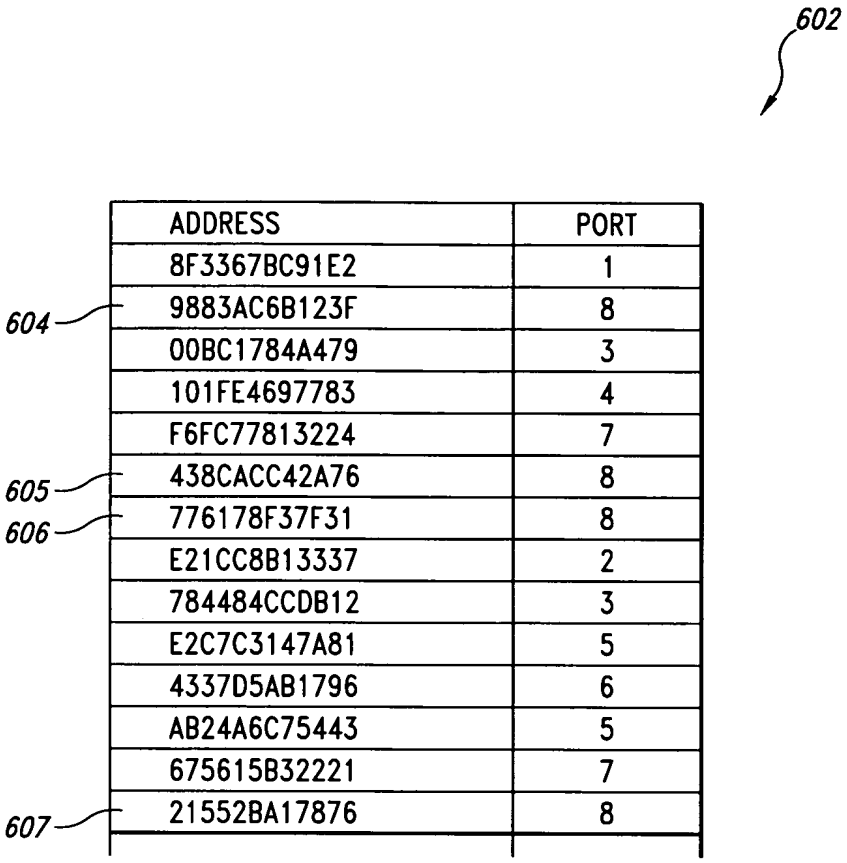
602
604
605
606
607
ADDRESS
PORT
8F3367BC91E2
1
9883AC6B123F
8
00BC1784A479
3
101FE4697783
4
F6FC77813224
7
438CACC42A76
8
776178F37F31
8
E21CC8B13337
2
784484CCDB12
3
E2C7C3147A81
5
4337D5AB1796
6
AB24A6C75443
5
675615B32221
7
21552BA17876
8

Fig. 6

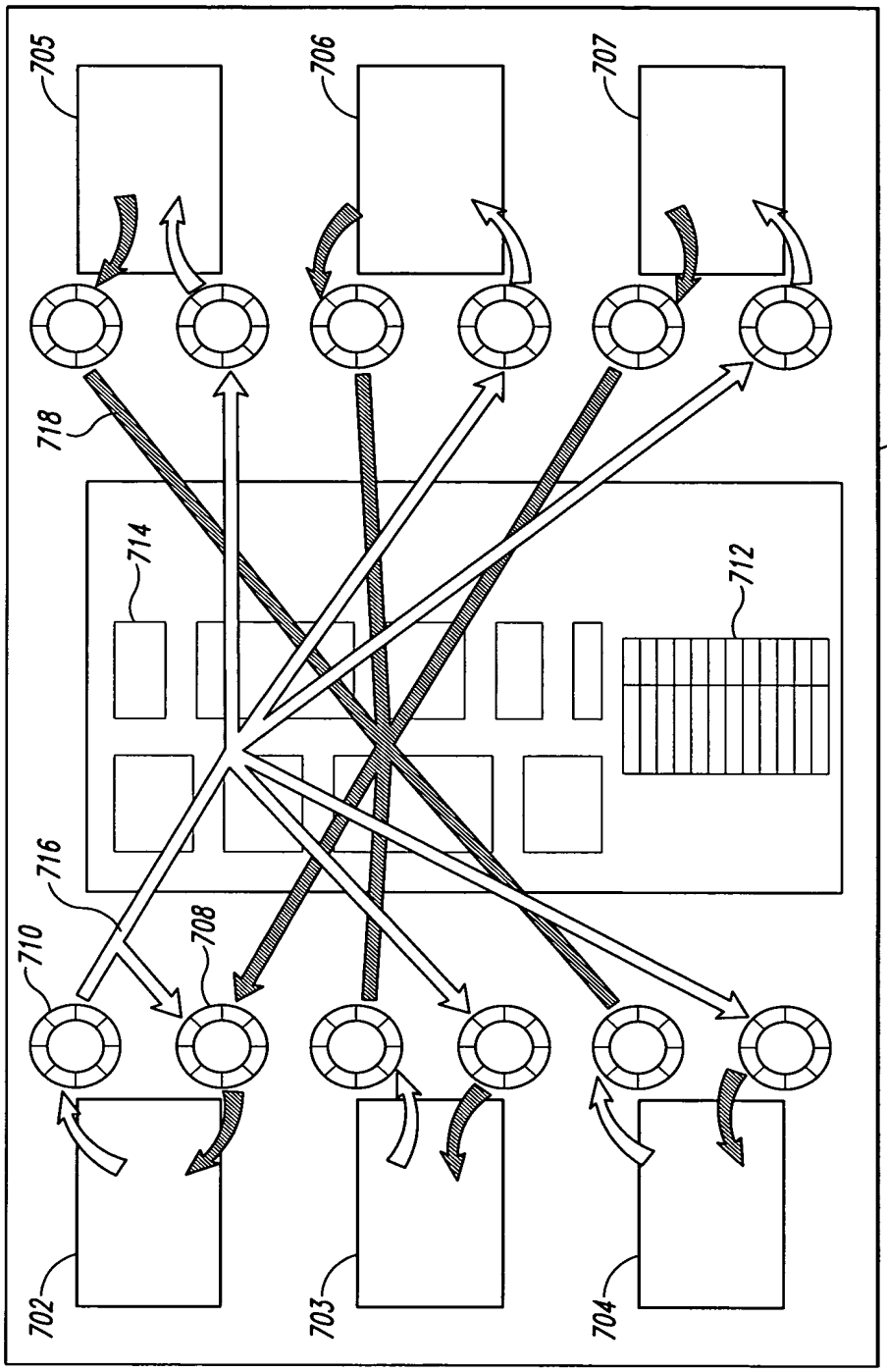

Fig. 7

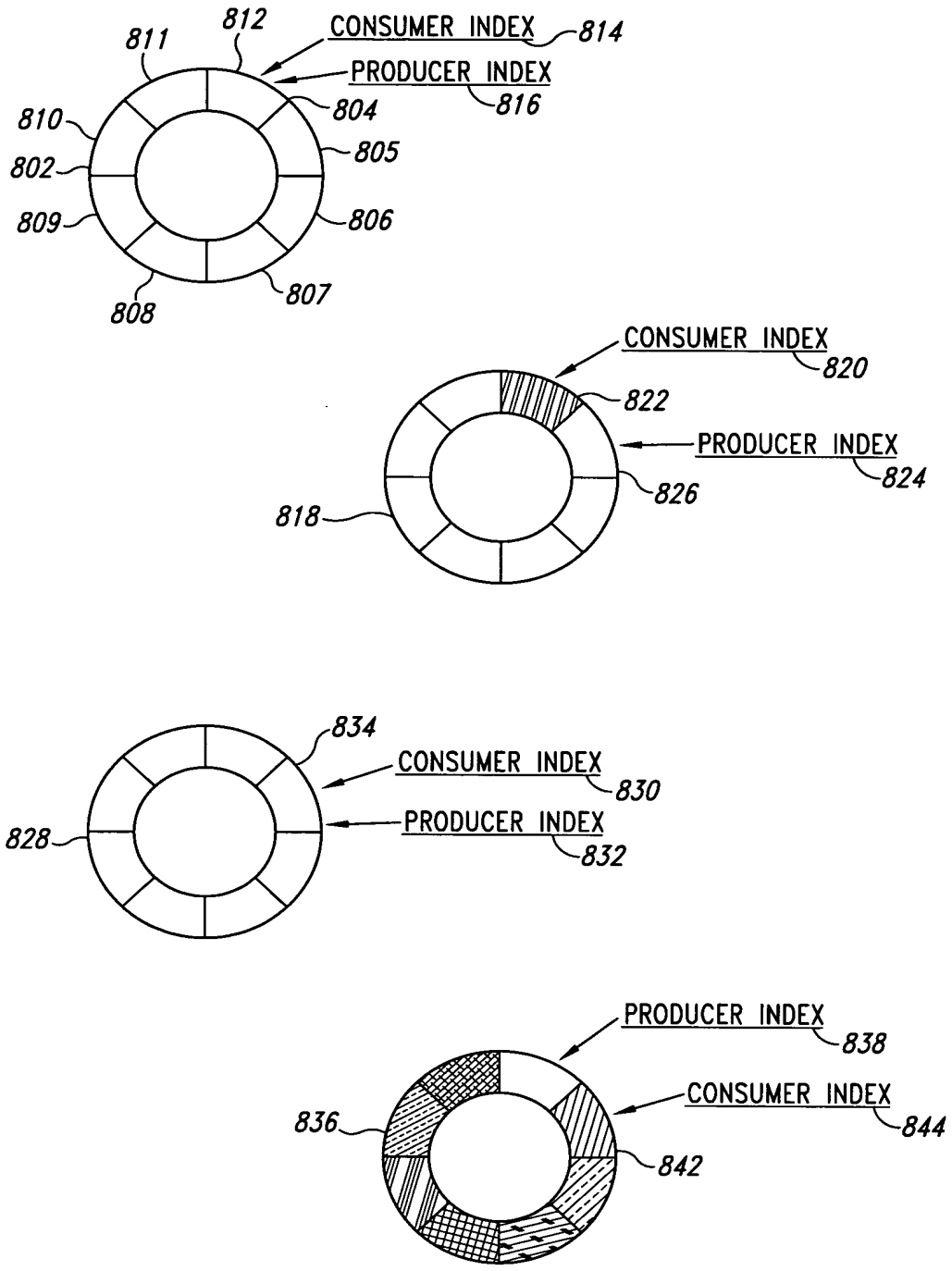
811
812
CONSUMER INDEX
814
PRODUCER INDEX
816
804
810
805
802
809
806
808
807
CONSUMER INDEX
820
822
PRODUCER INDEX
824
826
818
834
CONSUMER INDEX
830
828
PRODUCER INDEX
832
PRODUCER INDEX
838
CONSUMER INDEX
844
836
842

Fig. 8

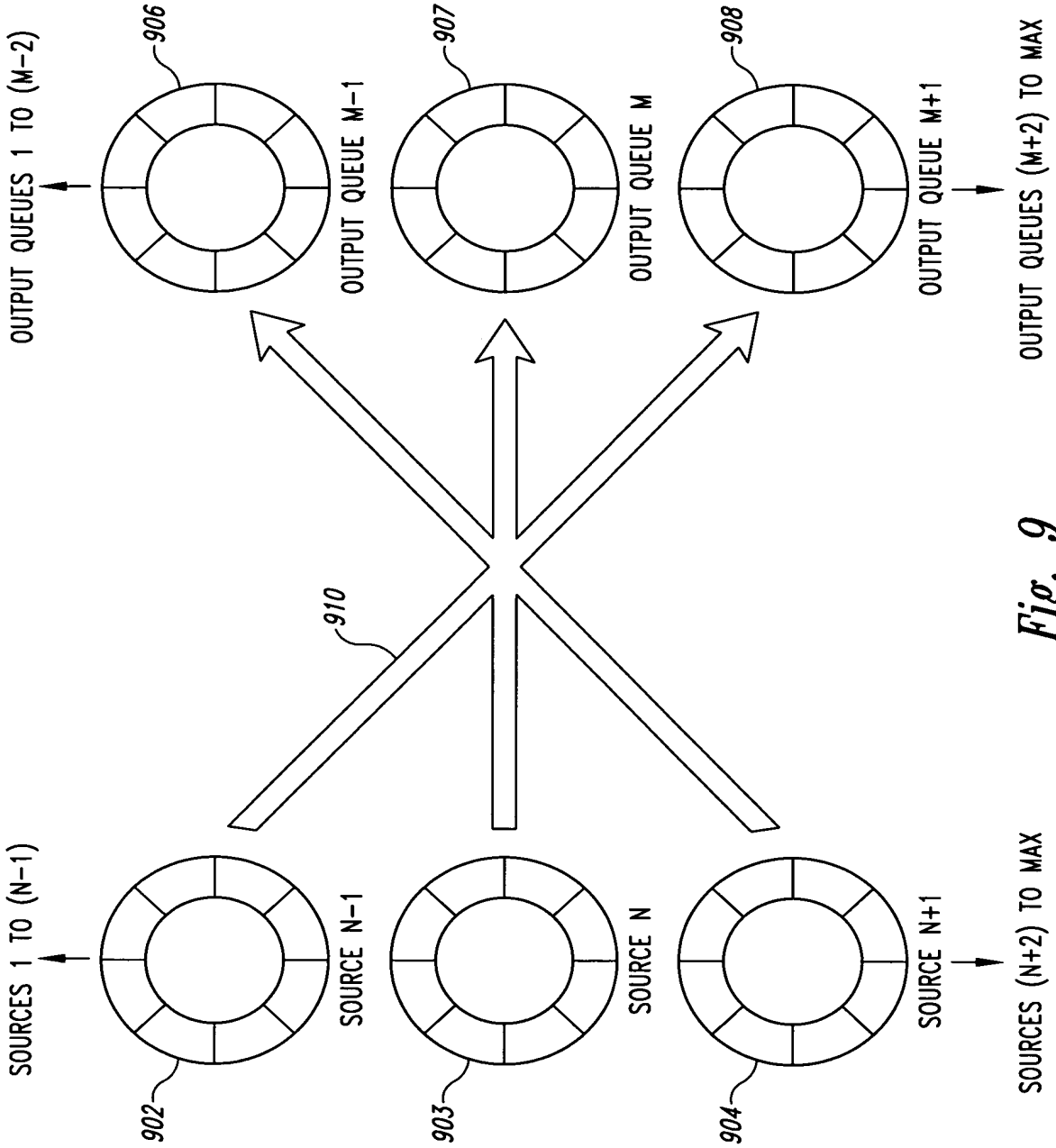
SOURCES 1 TO (N-1)
SOURCE N-1
SOURCE N
SOURCE N+1
SOURCES (N+2) TO MAX
902
903
904
910
OUTPUT QUEUES 1 TO (M-2)
OUTPUT QUEUE M-1
OUTPUT QUEUE M
OUTPUT QUEUE M+1
OUTPUT QUEUES (M+2) TO MAX
906
907
908

Fig. 9

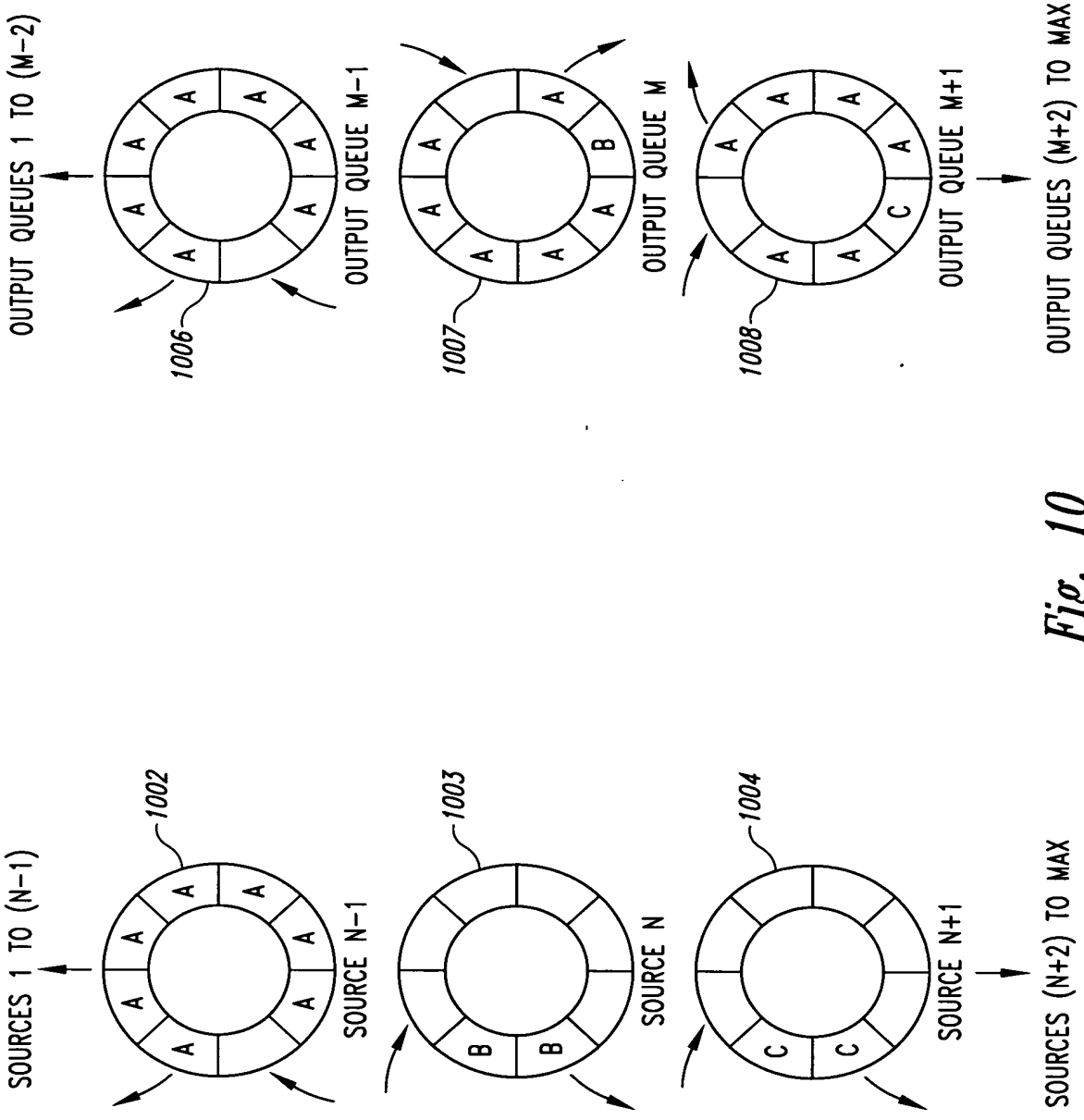
SOURCES 1 TO (N-1)
SOURCE N-1
SOURCE N
SOURCE N+1
SOURCES (N+2) TO MAX
OUTPUT QUEUES 1 TO (M-2)
OUTPUT QUEUE M-1
OUTPUT QUEUE M
OUTPUT QUEUE M+1
OUTPUT QUEUES (M+2) TO MAX
1002
1003
1004
1006
1007
1008

Fig. 10

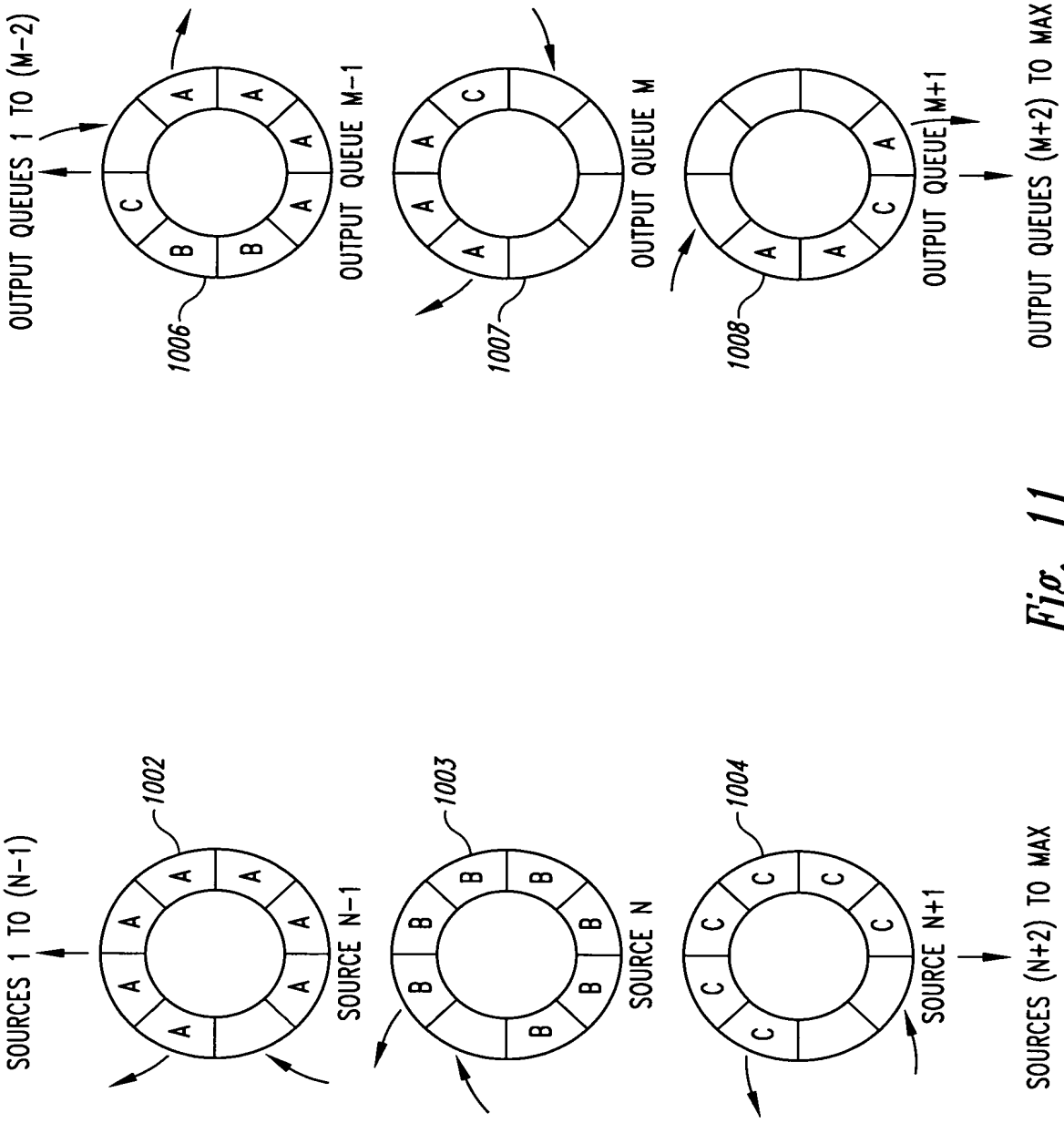
OUTPUT QUEUES 1 TO (M-2)
OUTPUT QUEUE M-1
OUTPUT QUEUE M
OUTPUT QUEUE M+1
OUTPUT QUEUES (M+2) TO MAX
1006
1007
1008
SOURCES 1 TO (N-1)
SOURCE N-1
SOURCE N
SOURCE N+1
SOURCES (N+2) TO MAX
1002
1003
1004

Fig. 11

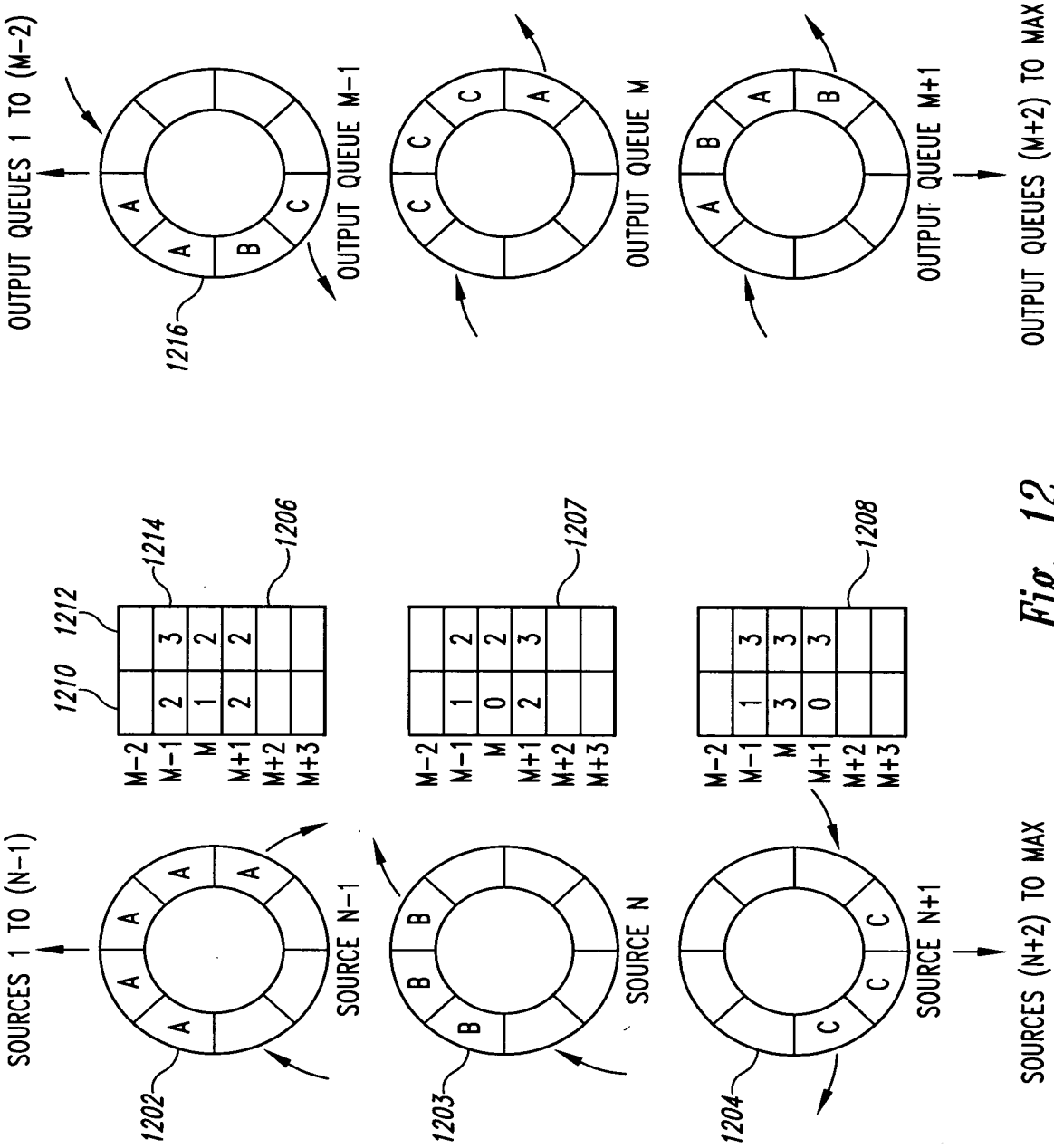
OUTPUT QUEUES 1 TO (M-2)
OUTPUT QUEUE M-1
OUTPUT QUEUE M
OUTPUT QUEUE M+1
OUTPUT QUEUES (M+2) TO MAX
SOURCES 1 TO (N-1)
SOURCE N-1
SOURCE N
SOURCE N+1
SOURCES (N+2) TO MAX
1202
1203
1204
1206
1207
1208
1210
1212
1214
1216
M-2
M-1
M
M+1
M+2
M+3

Fig. 12

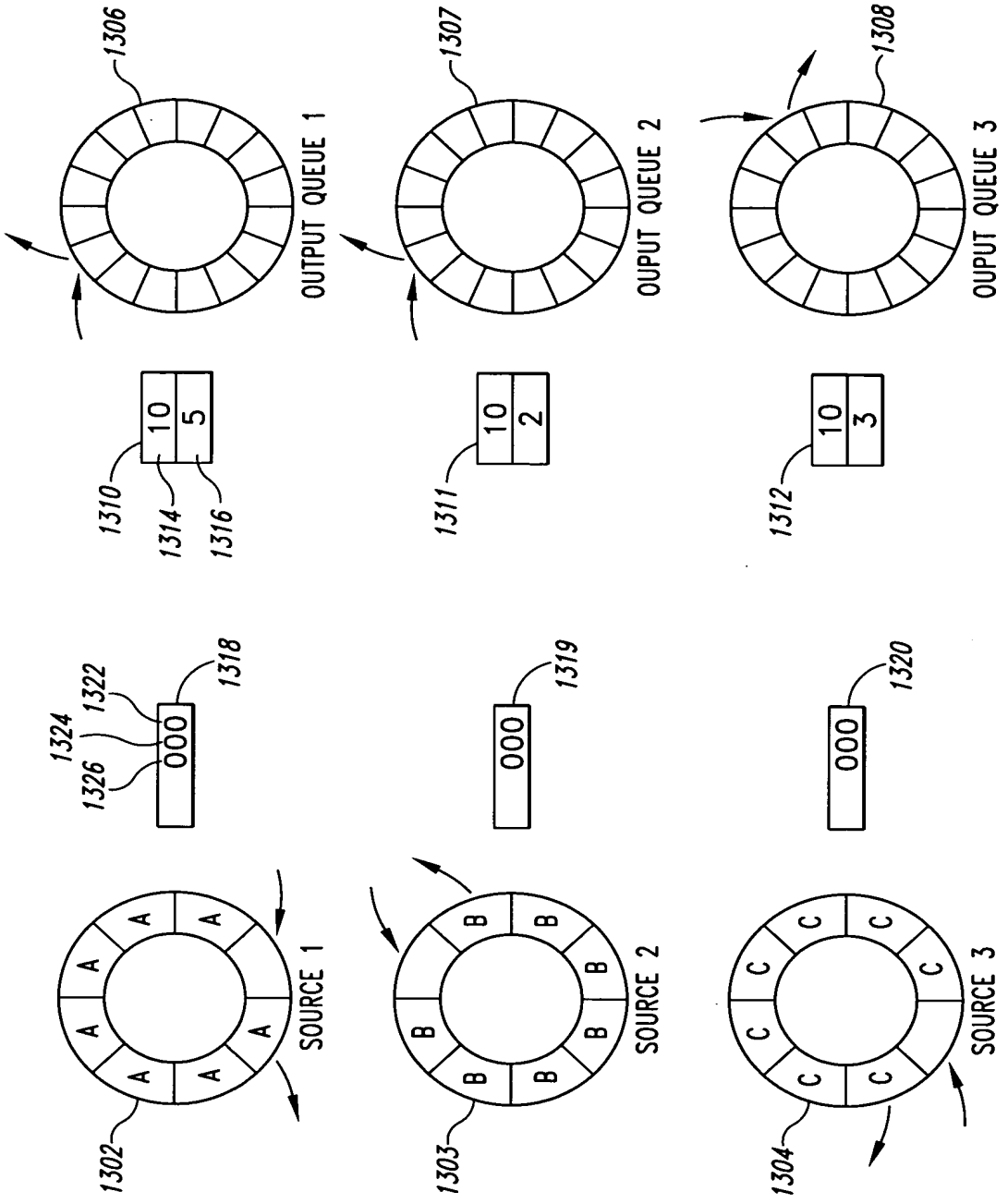
1306
OUTPUT QUEUE 1
1307
OUPUT QUEUE 2
1308
OUPUT QUEUE 3
10
5
10
2
10
3
1310
1314
1316
1311
1312
1324
1322
1326
000
1318
000
1319
000
1320
A
B
C
SOURCE 1
SOURCE 2
SOURCE 3
1302
1303
1304

Fig. 13A

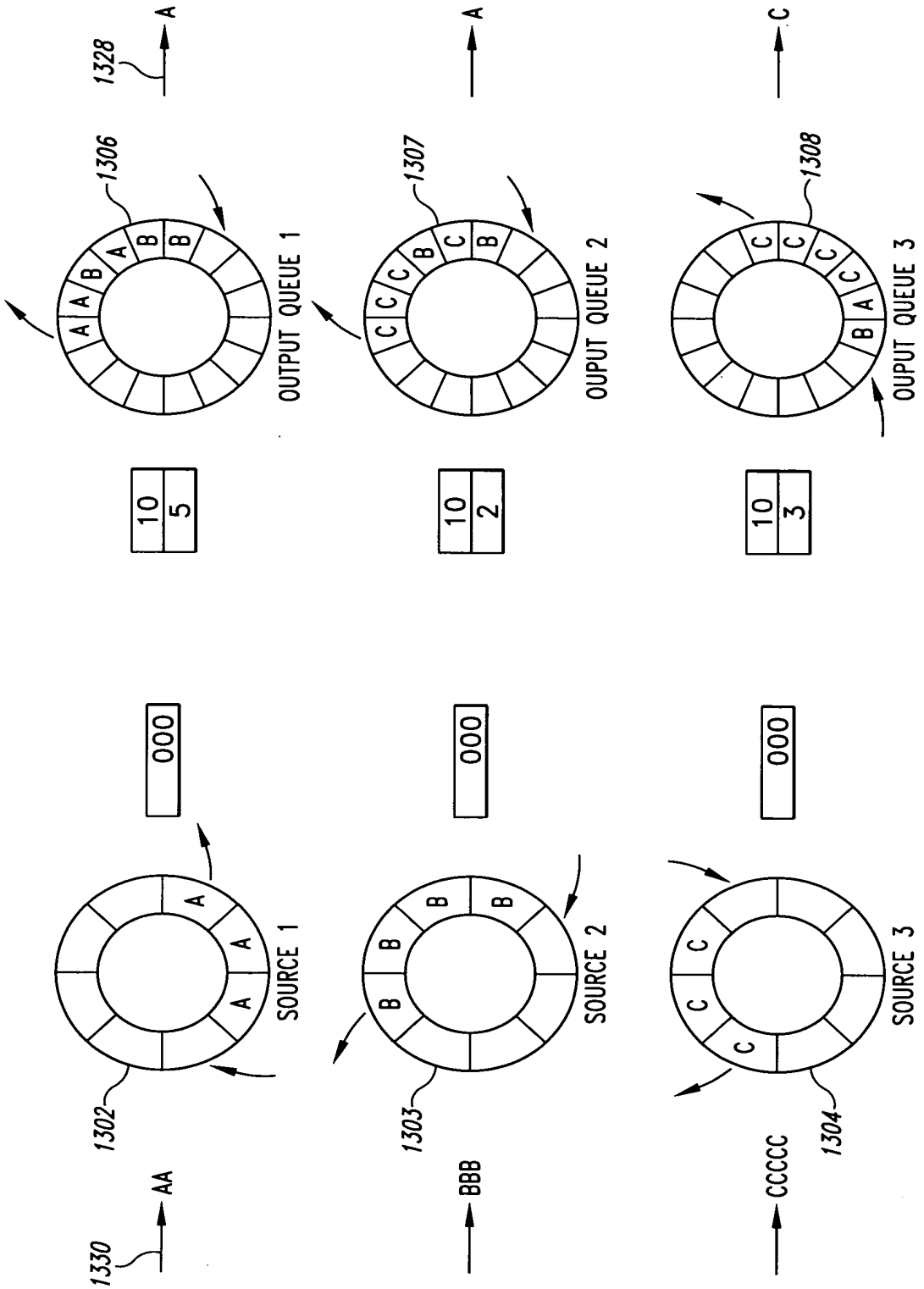
1330
AA
1302
SOURCE 1
000
10
5
1306
OUTPUT QUEUE 1
1328
A
BBB
1303
SOURCE 2
000
10
2
1307
OUPUT QUEUE 2
A
CCCCC
1304
SOURCE 3
000
10
3
1308
OUPUT QUEUE 3
C

Fig. 13B

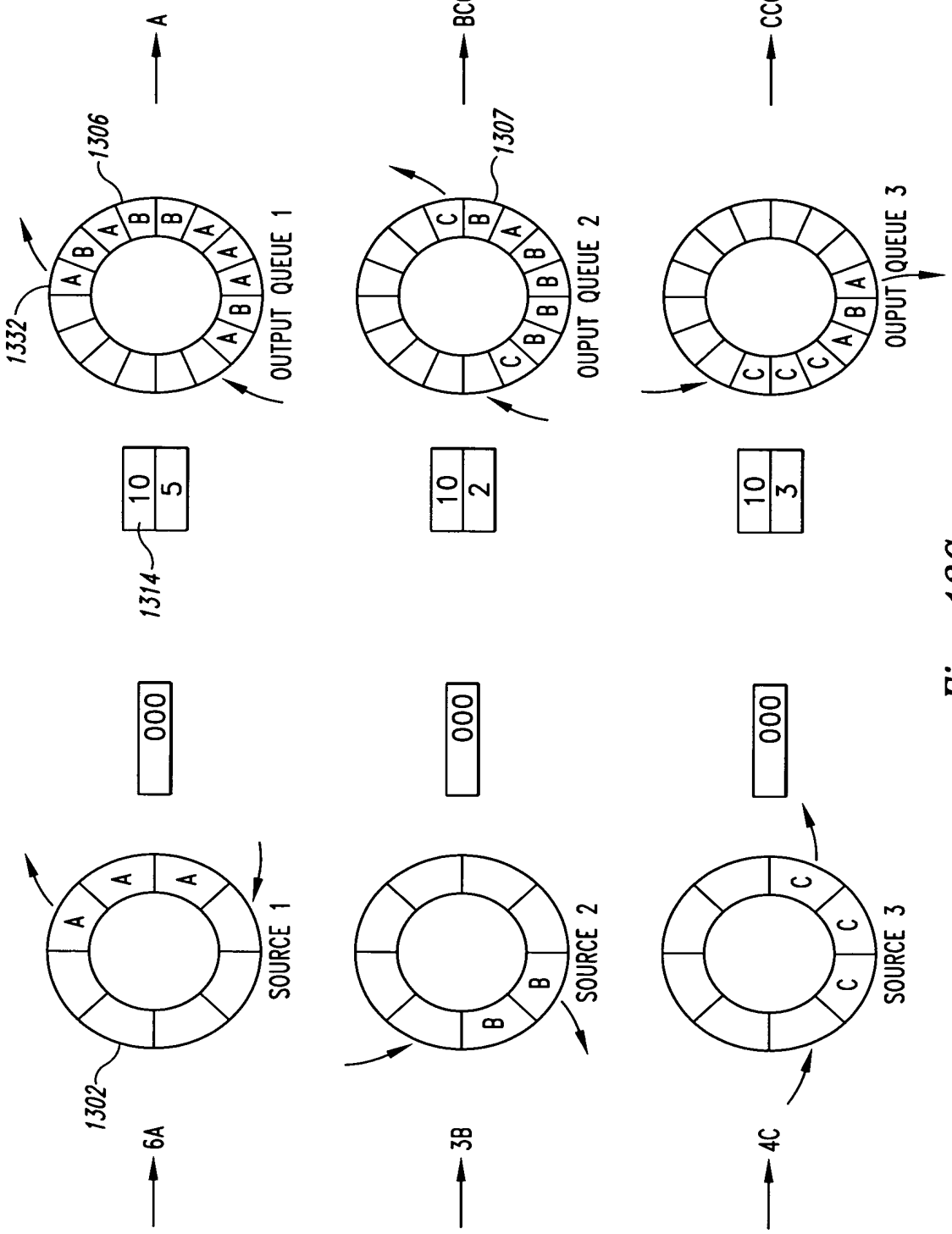
6A
3B
4C
1302
SOURCE 1
SOURCE 2
SOURCE 3
000
000
000
1314
10
5
10
2
10
3
1332
1306
1307
OUTPUT QUEUE 1
OUPUT QUEUE 2
OUPUT QUEUE 3
A
BCCC
CCCC

Fig. 13C

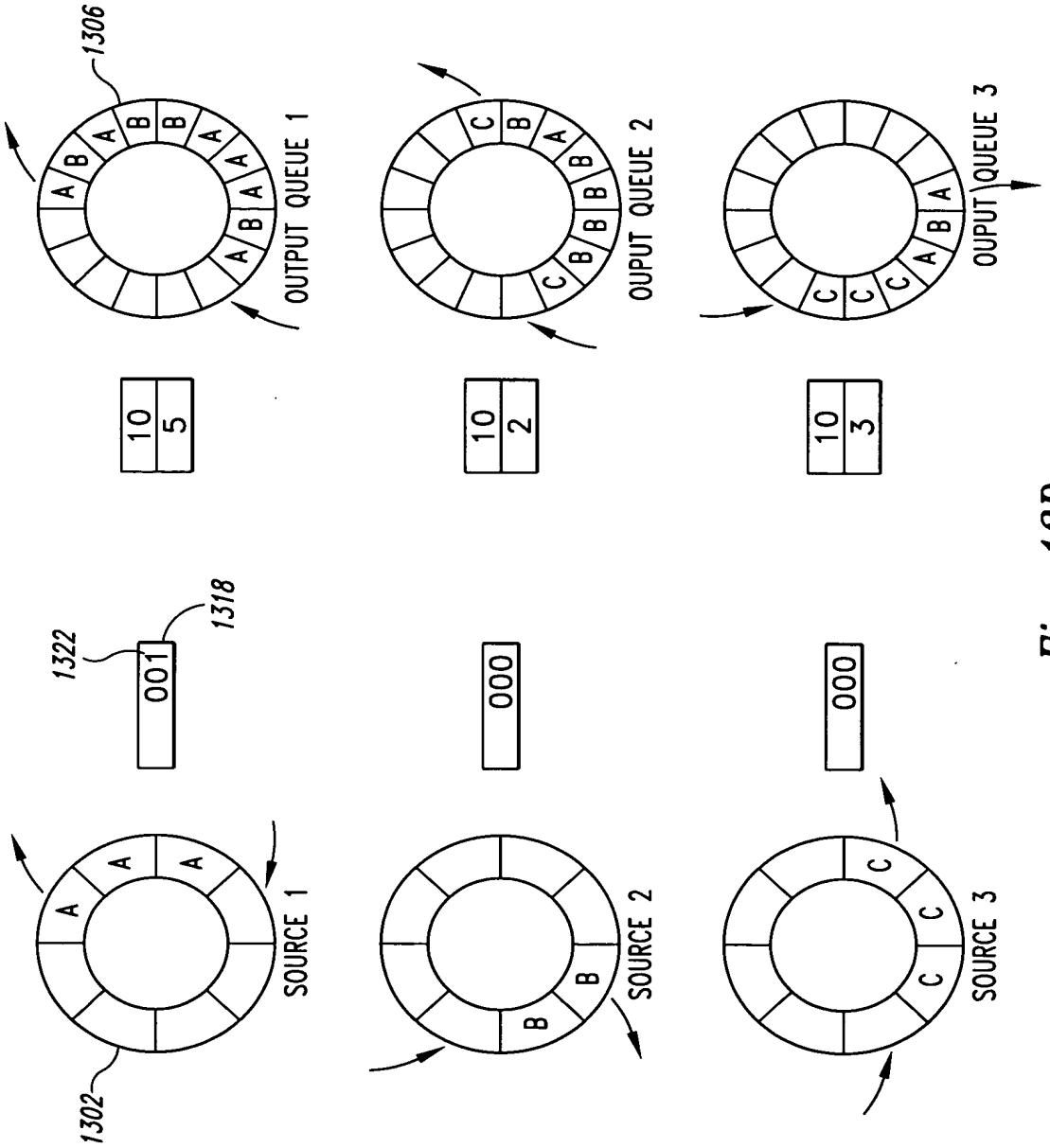
1306
OUTPUT QUEUE 1
OUPUT QUEUE 2
OUPUT QUEUE 3
10
5
10
2
10
3
1322
001
1318
000
000
SOURCE 1
SOURCE 2
SOURCE 3
1302

Fig. 13D

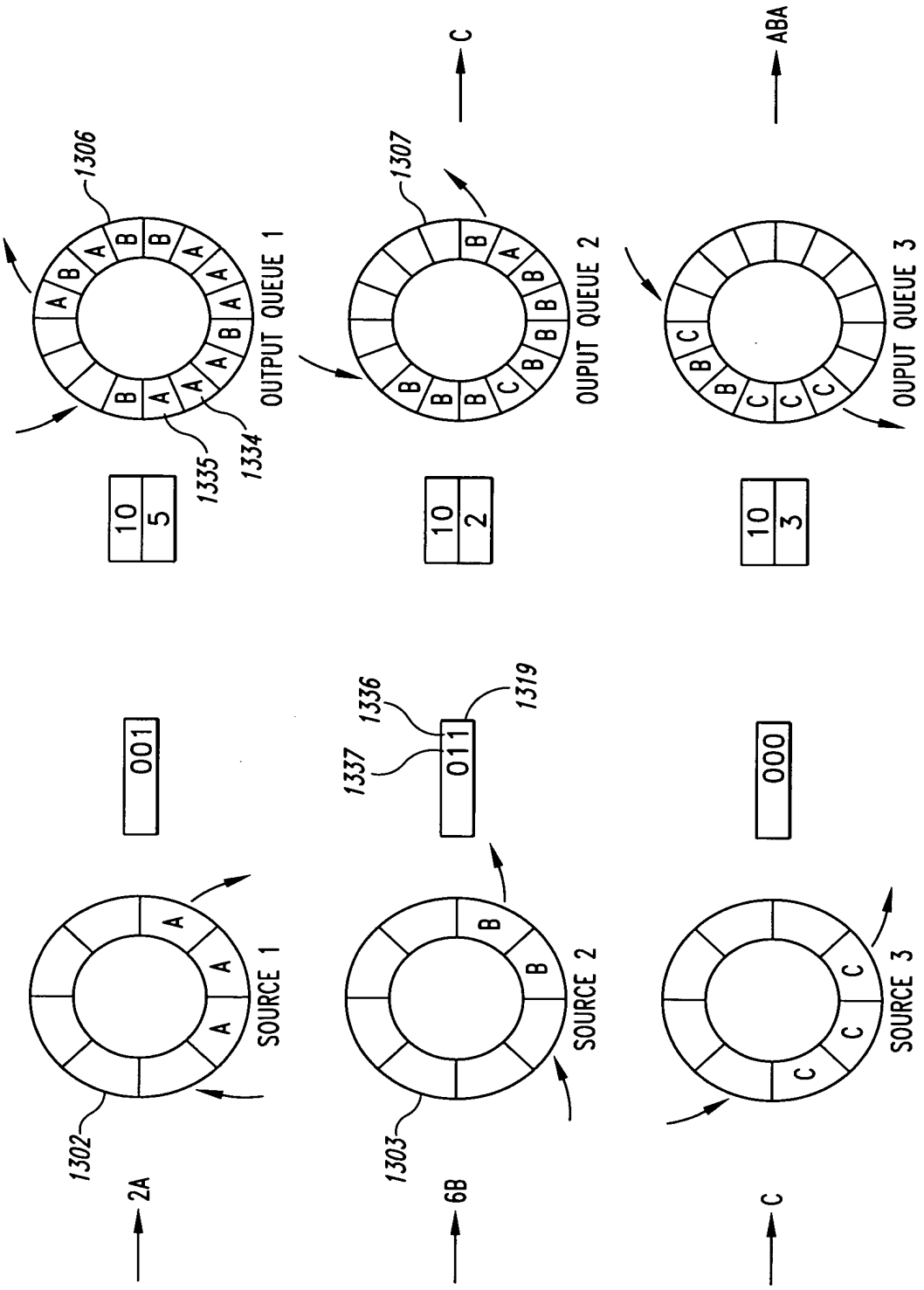
2A
1302
SOURCE 1
001
6B
1303
SOURCE 2
1337
1336
011
1319
C
SOURCE 3
000
1335
1334
OUTPUT QUEUE 1
1306
10
5
C
OUPUT QUEUE 2
1307
10
2
ABA
OUPUT QUEUE 3
10
3

Fig. 13E

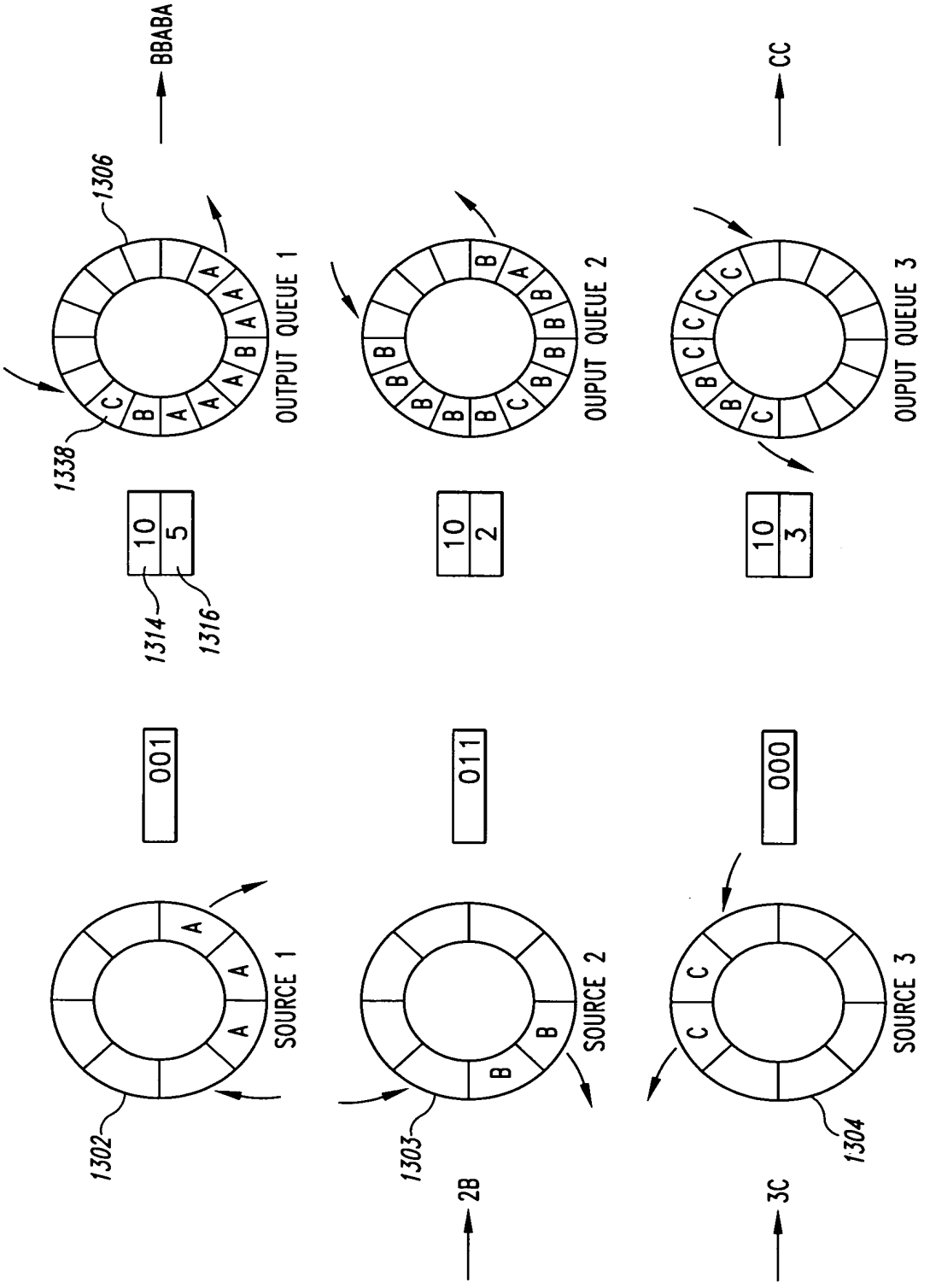
SOURCE 1
SOURCE 2
SOURCE 3
001
011
000
1302
1303
1304
2B
3C
10
5
10
2
10
3
1314
1316
1306
1338
OUTPUT QUEUE 1
OUPUT QUEUE 2
OUPUT QUEUE 3
BBABA
CC

Fig. 13F

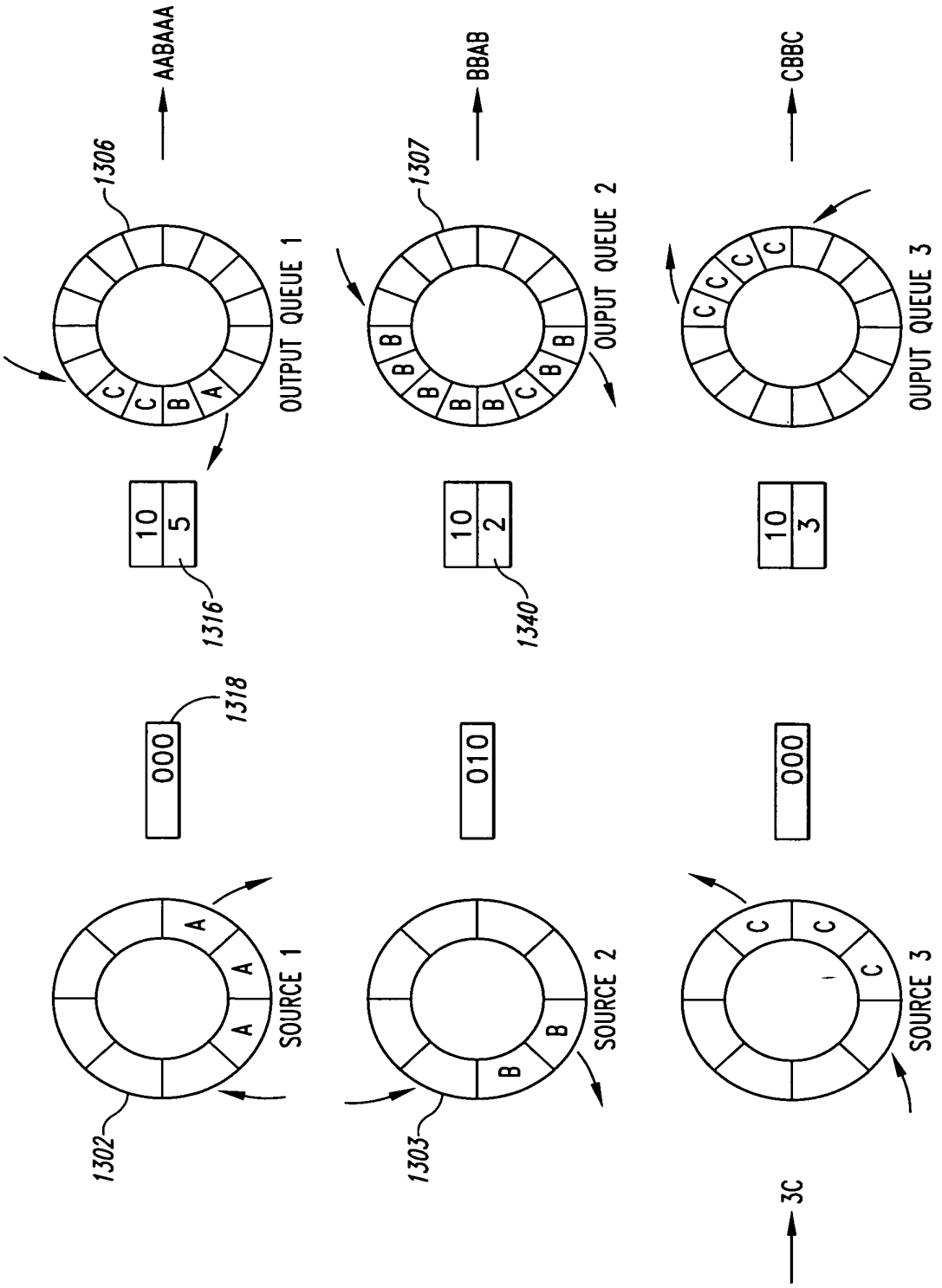
SOURCE 1
SOURCE 2
SOURCE 3
1302
1303
3C
000
010
000
1318
10
5
10
2
10
3
1316
1340
OUTPUT QUEUE 1
OUPUT QUEUE 2
OUPUT QUEUE 3
1306
1307
AABAAA
BBAB
CBBC

Fig. 13G

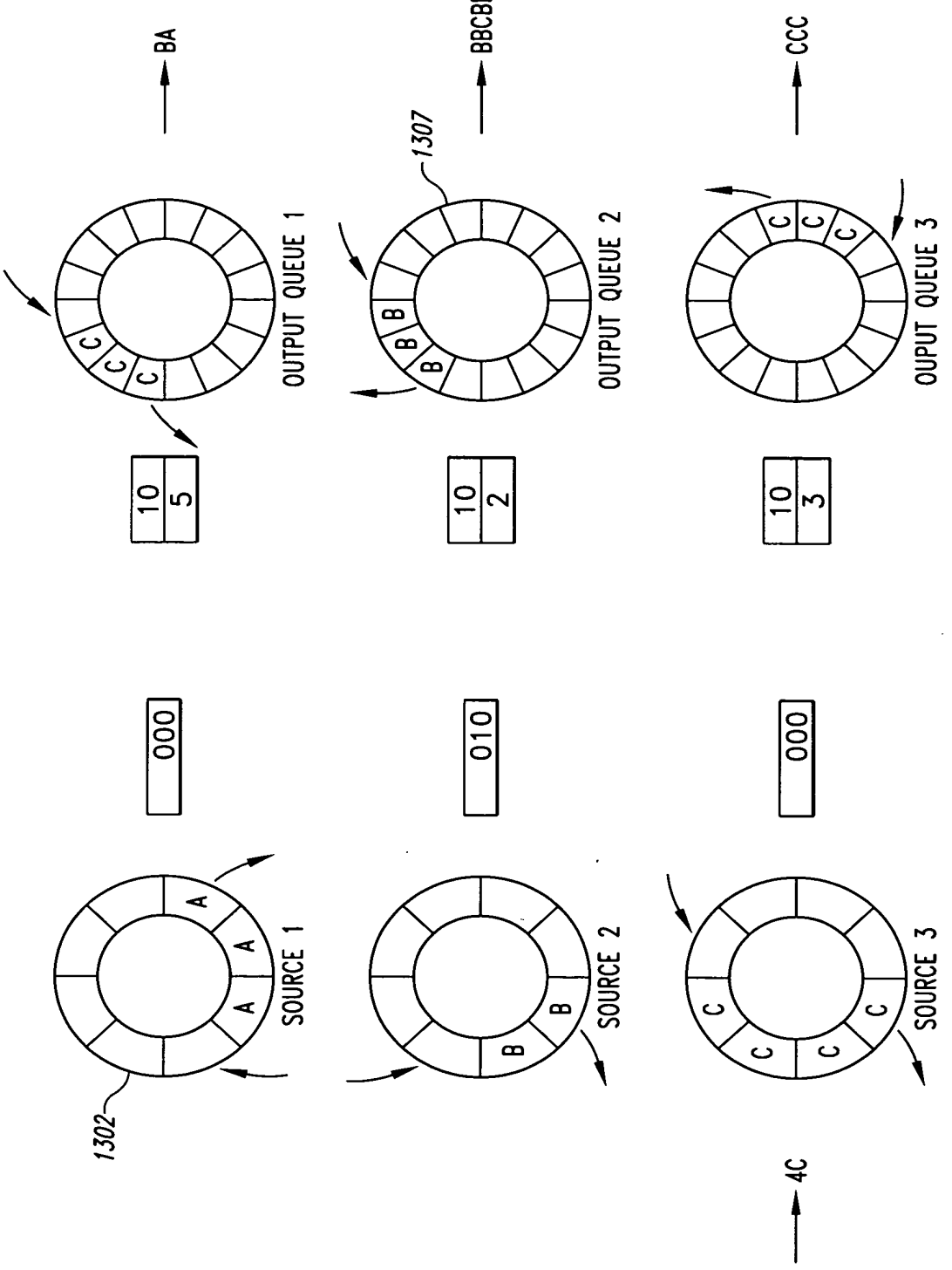
SOURCE 1
SOURCE 2
SOURCE 3
1302
000
010
000
10
5
10
2
10
3
OUTPUT QUEUE 1
OUTPUT QUEUE 2
OUPUT QUEUE 3
1307
BA
BBCBB
CCC
4C

Fig. 13H

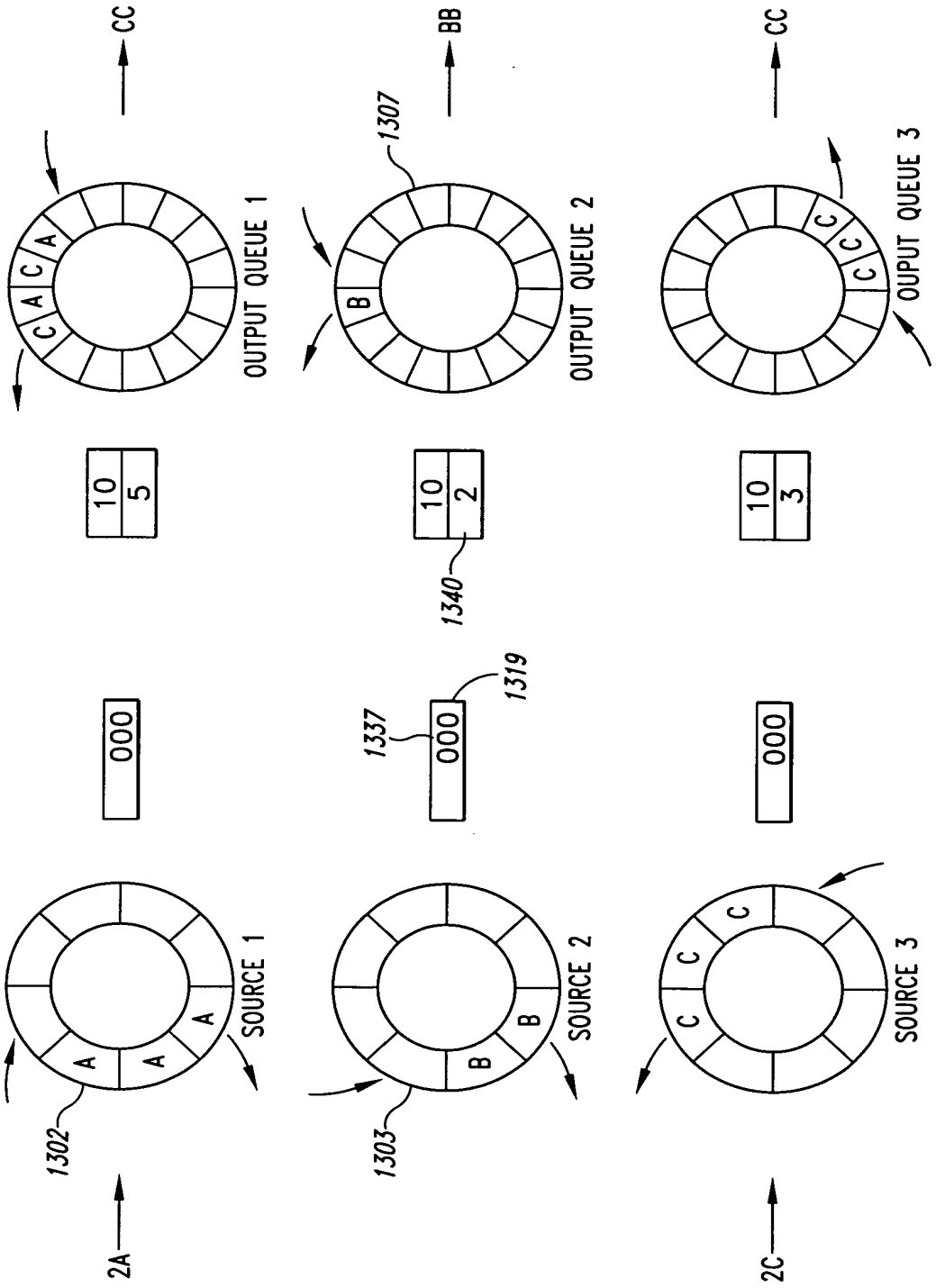
2A
2C
1302
1303
1337
1319
1340
1307
SOURCE 1
SOURCE 2
SOURCE 3
OUTPUT QUEUE 1
OUTPUT QUEUE 2
OUPUT QUEUE 3
CC
BB
CC
000
000
000
10
5
10
2
10
3
A
A
A
A
C
A
C
A
B
B
C
C
C
C
C

Fig. 13I

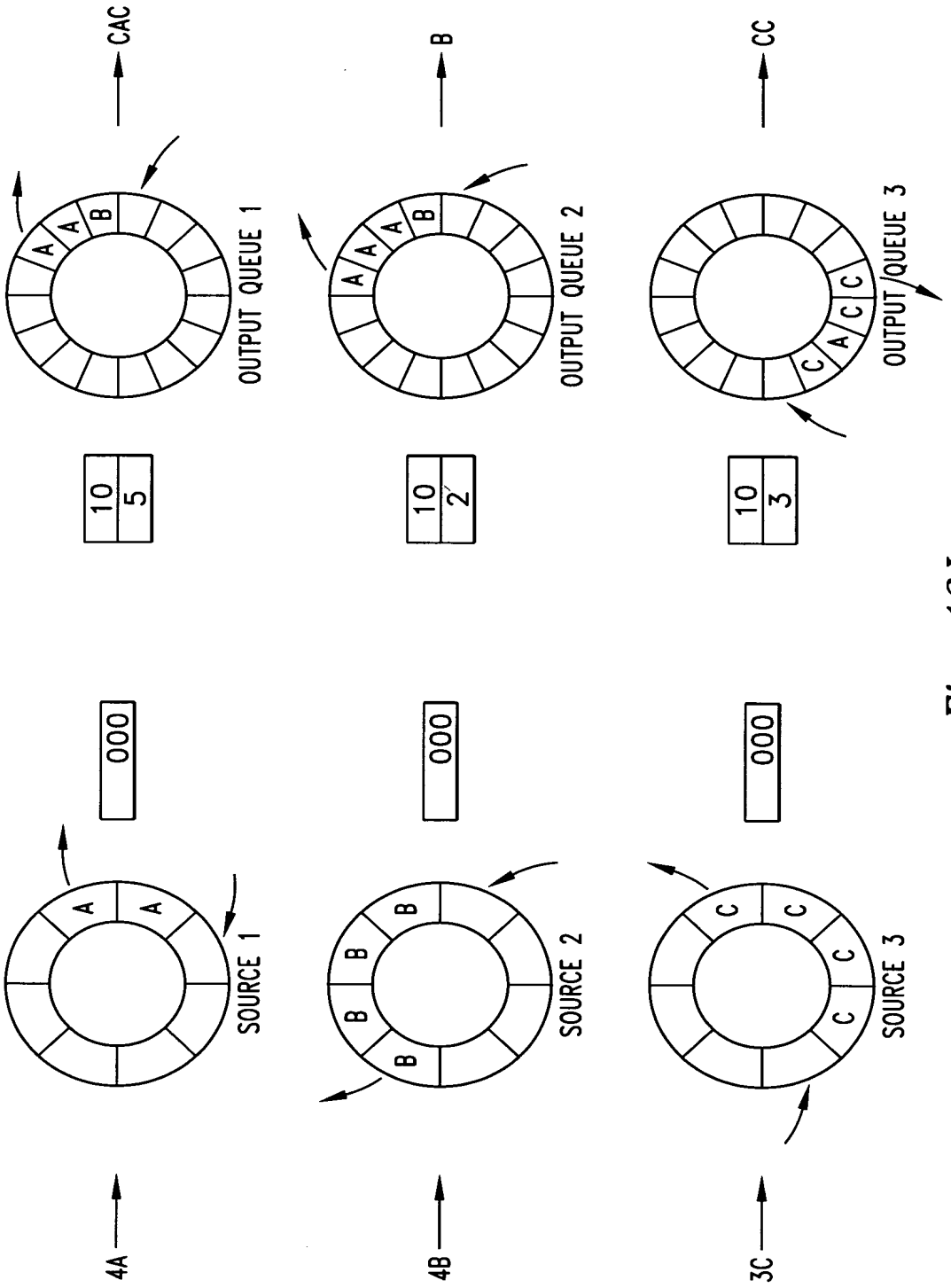
CAC
B
CC
OUTPUT QUEUE 1
OUTPUT QUEUE 2
OUTPUT QUEUE 3
10
5
10
2
10
3
000
000
000
SOURCE 1
SOURCE 2
SOURCE 3
4A
4B
3C

Fig. 13J

```
1 const FORWARD_AFTER_CONTROL=2;
2 typedef void (*callbackFunction)();
3
4 class message
5 {
6    private:
7           char msg[1000];
8           int msgNo;
9    public:
10          int  getLength();
11          int  getSource();
12          int  getDestination();
13          char* getMessage();
14          void  setNo (int no);
15          int  getNo();
16          message();
17          ~message();
18 };
19
20 class messageDescriptor
21 {
22    private:
23          message* msgPtr;
24          bool   flag;
25
26    public:
27          message* getMsg();
28          void   setMsg(message *m);
29          void   zeroFlag();
30          bool   getFlag();
31          bool   query();
32          messageDescriptor();
33          ~messageDescriptor();
34 };
35
```

Fig. 14A

```
36 class cQueue
37 {
38    private:
39           messageDescriptor queue[QUEUE_SIZE];
40           int head;
41           int tail;
42           bool inc(int & ptr);
43           bool dec(int & ptr);
44
45    public:
46           bool empty();
47           bool full();
48           int  num();
49           bool queueMDesc (messageDescriptor & msg);
50           bool dequeueMDesc (messageDescriptor & msg);
51           cQueue();
52           virtual ~cQueue();
53 };
54
55 class transceiver
56 {
57    public:
58
59           flowControl();
60           releaseFlowControl();
61           kick();
62           startTransceiver(cQueue *out, callbackFunction signalSend,
63                            cQueue *in, callbackFunction signalReceive);
64           transceiver();
65           ~transceiver();
66 };
67
68 class port;
69
70 class portList
71 {
72    public:
73           addPort (port *p);
74           removePort(port *p);
75           port* firstPort();
76           port* nextPort();
77           bool in(port *p);
78           bool empty();
79           bool clear();
80 };
81
```

Fig. 14B

```
 82 class portDirectory
 83 {
 84    public:
 85           port *getPortFromSource(int sid);
 86 };
 87
 88 class port
 89 {
 90    private:
 91           cQueue receiveQueue;
 92           cQueue transmitQueue;
 93           bool overflow;
 94           bool flowControlled;
 95           int highThreshold;
 96           int lowThreshold;
 97           portList controlled;
 98           portList controlledBy;
 99           portDirectory* portDir;
100           transceiver portHardware;
101
102    public:
103           void receiveMessage(messageDescriptor m, port *sPort);
104           void incomingMessage();
105           void outgoingMessage();
106           void receiveFlowControl(port *sPort);
107           void receiveFlowControlRelease(port *sPort);
108           port(portDirectory *p, int low, int high);
109           ~port();
110 };
```

Fig. 14C

```
1 void port::receiveMessage(messageDescriptor m, port *sPort)
2 {
3     if (transmitQueue.num() > highThreshold - 2)
4     {
5             if (!controlled.in(sPort))
6             {
7                     sPort->receiveFlowControl(this);
8                     controlled.addPort(sPort);
9             }
10            overflow = TRUE;
11    }
12    if (!transmitQueue.full())
13    {
14            transmitQueue.queueMDesc(m);
15            portHardware.kick();
16    }
17 }
18
19 void port::incomingMessage()
20 {
21    messageDescriptor m;
22    port *p;
23
24    if (controlledBy.empty())
25    {
26            if (flowControlled)
27            {
28                    flowControlled = FALSE;
29                    portHardware.releaseFlowControl();
30            }
31            while (!receiveQueue.empty() && !flowControlled)
32            {
33                    receiveQueue.dequeueMDesc(m);
34                    p = portDir->getPortFromSource((m.getMsg())->getSource());
35                    p->receiveMessage(m, p);
36            }
37            portHardware.kick();
38    }
39 }
40
```

Fig. 14D

```
41 void port::outgoingMessage()
42 {
43     port* nxt;
44
45     if (overflow && transmitQueue.num() < lowThreshold)
46     {
47             overflow = FALSE;
48             nxt = controlled.firstPort();
49             while (nxt)
50             {
51                     nxt->receiveFlowControlRelease(this);
52                     nxt = controlled.nextPort();
53             }
54             controlled.clear();
55     }
56 }
57
58 void port::receiveFlowControl(port *sPort)
59 {
60     int num = FORWARD_AFTER_CONTROL;
61     messageDescriptor m;
62     port *p;
63
64     controlledBy.addPort(sPort);
65     portHardware.flowControl();
66
67     //optional
68     while (num-- && !receiveQueue.empty())
69     {
70             receiveQueue.dequeueMDesc(m);
71             p = portDir->getPortFromSource((m.getMsg())->getSource());
72             p->receiveMessage(m, p);
73     }
74 }
75
```

Fig. 14E

```
76 void port::receiveFlowControlRelease(port *sPort)
77 {
78     controlledBy.removePort(sPort);
79     incomingMessage();
80 }
81 port::port(portDirectory *p, int low, int high)
82 {
83             cQueue* t = &(transmitQueue);
84             cQueue* r = &(receiveQueue);
85             overflow = FALSE;
86             flowControlled = FALSE;
87             highThreshold = high;
88             lowThreshold = low;
89             portDir = p;
90             portHardware.startTransceiver(t, port::outgoingMessage(),
91                                           r, port::incomingMessage());
92 }
```

Fig. 14F